Devoir en temps limité n°1 – 2h

Calculatrices autorisées

On veillera à présenter très clairement sa copie : il faut rédiger les réponses et encadrer les résultats. Pour le code, il doit être indenté, on ne commence pas une fonction en bas de page et on utilise de la couleur pour les commentaires.

Le code doit être commenté dès qu'il dépasse les 5 lignes.

Le code C demandé, sauf demande explicite, doit être écrit dans une fonction et pas dans le `main`. On supposera que toutes les bibliothèques nécessaires ont été importées.

1 Un peu de C

1. Rappeler les deux lignes permettant la compilation et l'exécution d'un code C. On supposera que notre code est dans un fichier nommé **`fichier.c`**.
2. Écrire une fonction `float maximum(float* tab, int n)` qui calcule le maximum du tableau **`tab`**.
3. Pour chacune des versions suivantes de la fonction f, calculer la valeur renvoyée.

```c
int f1 () {
   int res = 0;
   for(int i=0;i<6;i+=1){
     res+=i;
   }
   return res;
}
```

```c
int f2 () {
   int res = 0;
   for(int i=0;i<=6;i+=1){
     res+=i;
   }
   return res;
}
```

```c
int f3 () {
   int res = 0;
   for(int i=6;i>0;i-=1){
     res+=i;
   }
   return res;
}
```

```c
int f4 () {
   int res = 0;
   for(int i=6; i>=0; i+=1){
     res+=i;
   }
   return res;
}
```

4. Écrire une fonction `int somme(int n)` qui calcule la somme $\sum_{k=0}^{n} k^2 + 3$.
5. Écrire une fonction `int somme_puissances(int n, int x)` qui calcule la somme $\sum_{k=0}^{n} x^k$
6. On définit la suite $(u_n)_{n\in\mathbb{N}}$ suivante : $u_0 = 0$, $u_1 = 3$ et $\forall n \geq 2$, $u_n = 3 * u_{n-1} + 10 * u_{n-2} + 2$.
   Écrire une fonction **récursive** `int calcul_u(int n)` qui calcule le n-ième terme de la suite.
7. Dessiner l'état de la mémoire après les lignes de code suivant (on supposera être dans le `main`) :

```c
int* p = malloc(sizeof(int));
*p = 3;
int x = 0;
int* q = &x;
q=p;
```

8. Dessiner une représentation des différents appels récursifs réalisés par la fonction suivante lors de l'appel `pgcd(72,27)`. On précisera la valeur de retour de chaque appel.

```c
int pgcd(int a, int b){
  //La suite du code marche uniquement si a>b. Si ce n'est pas le cas, on échange les entrées
  if(a < b){return pgcd(b,a);}

  if (a%b==0){return b;}
  else{return pgcd(b, a%b);}
}
```

9. Écrire une fonction `float moyenne_sans_0(float* notes, int n)` qui prend en entrée un tableau de n notes et fait la moyenne des notes, en ignorant les notes valant 0 (absence au devoir).
   Par exemple, pour $notes = [12, 9, 14, 0, 15]$ et $n = 5$, la fonction doit renvoyer $50/4 = 12.5$.

2 Un peu de binaire

10. Écrire les nombres suivants en binaire : 12, 142, 269.
11. Que représentent les écritures binaires suivantes : 1011, 1001010 ?
12. Effectuer les calculs suivants en binaire (en écrivant les étapes suivies et en posant les calculs) : 30-23, 100+12. On fera les calculs sur un octet.
13. Quel est le plus grand nombre qu'on peut représenter en binaire sur 2 octets ? Sur 4 octets ? (ici on parle bien de binaire classique, pas de la représentation des entiers relatifs)
14. Soit $n \in \mathbb{N}$. Combien de chiffres en binaire (ou bits) faut-il pour représenter 2^n ? Pour représenter $2^n - 1$?

3 Un peu de bash

Le but de cet exercice est d'utiliser quelques commandes de bash. On se place à la racine `/`.

15. Se ramener dans votre répertoire personnel.
16. Créer un nouveau dossier **`tp`**.
17. Aller dans le dossier **`tp`**.
18. Quelle commande permet d'afficher le contenu du dossier ?
19. Créer un nouveau fichier **`tp.c`**

On suppose que vous avez écrit du code dans le fichier et l'avez compilé en un fichier **`tp.out`**.
Lorsque vous essayez d'exécuter, le terminal vous réponds que le fichier **`tp.out`** n'est pas exécutable.

20. Quelle commande faut il utiliser pour vérifier que notre fichier a bien les droits d'exécution ?

Le résultat est le suivant :

```
-rw-r--r-- 1 vous users 1690 4 octobre 2025 tp.c
-rw-r--r-- 1 vous users 1690 4 octobre 2025 tp.out
```

21. Changer les permissions pour que vous puissiez exécuter **`tp.out`**.
22. Modifier les permissions pour que tout le groupe **`users`** (votre groupe) puisse écrire dans le fichier **`tp.c`**.

4 Un peu de preuves de programmes

23. Qu'est-ce qu'un variant de boucle ?
24. Qu'est-ce qu'un invariant de boucle ?

On considère le code suivant, qui prend en entrée un tableau d'entiers, sa taille et un entier x :

```
int mystere(int* tab, int n, int x){
 int res = 0;
 int i = 0;
 while(i<n){
   if(tab[i]==x){res+=1;}
   i+=1;
 }

 return res;
}
```

25. Soit **`tab`** $= [1, 2, 1, 1, 3, 7, 1, 1]$, $n = 8$ et $x = 1$. Que renvoie la fonction pour ces entrées ?
26. Écrire sa spécification.
27. Déterminer et prouver un variant de la boucle while.
28. Qu'est-ce que la question précédente permet de conclure ?
29. Quel invariant permettrait de prouver la correction de la fonction ? Il n'est pas nécessaire de prouver que c'est bien un invariant.

5 Un peu de tri

Dans cette section on va étudier un algorithme de tri de tableaux qu'on appellera tri par positions.

On considère un tableau `t` de taille `n`. Le principe est le suivant :

- On crée un deuxième tableau `t2` de même taille que `t`.
- Pour chaque élément `t[i]` du tableau :
  - on détermine son rang `r` dans le tableau trié en comptant le nombre d'éléments qui lui sont strictement inférieurs.
  - Puis on le place dans la case `r` de `t2`.
- Le tableau `t2` est alors une version triée du tableau initial.

Dans un premier temps, on suppose qu'aucun élément n'est un doublon, c'est à dire que pour $i \neq j$ des indices du tableau, `t[i]`$\neq$`t[j]`.

30. Écrire une fonction `int compte_plus_petit(int* t, int n, int i)` qui prend en entrée un tableau, sa taille et un indice i du tableau et renvoie le nombre de valeurs dans le tableau qui sont strictement inférieures à `t[i]`.
31. Écrire une fonction `int* tri_position(int* t, int n)` qui renvoie un nouveau tableau qui contient les mêmes valeurs que t, mais triées dans l'ordre croissant. On utilisera la méthode décrite plus haut.

Dans un deuxième temps, on va généraliser au cas où le tableau peut contenir des doublons.

32. Montrer sur un exemple pourquoi le fait d'avoir un doublon pose problème avec notre méthode.
33. En utilisant des phrases et des schémas, expliquer une manière de résoudre le problème.
34. Programmer votre méthode. On peut réécrire entièrement la fonction `tri_position` ou écrire un bout de code à rajouter à la fonction (en précisant où on le rajoute).

Dans un troisième temps, on veut modifier notre tri pour qu'il soit **en place**. Cela signifie qu'on a plus le droit d'utiliser un deuxième tableau `t2` et on doit modifier le tableau initial.

35. (Difficile) Écrire une fonction `void tri_position_en_place(int* t, int n)` qui trie le tableau par positions, mais sans utiliser d'autres tableaux que t.

Dans un dernier temps, on pourrait s'intéresser à la preuve de notre programme et montrer sa correction. Une autre question intéressante serait l'efficacité de ce tri par rapport à d'autres : on se posera ce genre de questions dans le chapitre de complexité qui arrive bientôt.